## 1. PostgreSQL JOIN Performance

I used the Neo4J flights data to create the flights database in PostgreSQL including the following tables: flights, airlines, airports. This simple data model was used in this performance comparison. Given that the data contained less than 30 rows, this performance test does not represent a real-world scenario.

The following SQL SELECT statement was composed for the performance comparison. The tables are FOREIGN KEY’d on their PRIMARY KEY’d id columns, but on index is present on the flights table’s columns.

|  |
| --- |
| SELECT a.name AS AirlineName, d.city AS Departed, o.city AS Arrived, f.\* FROM flights f  INNER JOIN airlines a ON a.id = f.airlineid  INNER JOIN airports d ON d.id = f.depart\_airportid  INNER JOIN airports o ON o.id = f.arrive\_airportid; |

|  |  |
| --- | --- |
| # | Execution Time |
| 1 | 32 ms |
| 2 | 32 ms |
| 3 | 31 ms |
| 4 | 31 ms |
| 5 | 31 ms |

Next, the following index was added to the flights.airlineid column:

|  |
| --- |
| CREATE INDEX ndx\_airlineid ON flights (airlineid ASC); |

Again the SQL SELECT statement from above was executed and execution time was measured.

|  |  |
| --- | --- |
| # | Execution Time |
| 1 | 16 ms |
| 2 | 16 ms |
| 3 | 31 ms |
| 4 | 16 ms |
| 5 | 31 ms |

Next, the following index was added to the flights.depart.airportid column:

|  |
| --- |
| CREATE INDEX ndx\_depart\_airportid ON flights (airlineid ASC); |

Again the SQL SELECT statement from above was executed and execution time was measured.

|  |  |
| --- | --- |
| # | Execution Time |
| 1 | 32 ms |
| 2 | 31 ms |
| 3 | 31 ms |
| 4 | 15 ms |
| 5 | 31 ms |

Next, the following index was added to the flights.arrive.airportid column:

|  |
| --- |
| CREATE INDEX ndx\_arrive\_airportid ON flights (airlineid ASC); |

Again the SQL SELECT statement from above was executed and execution time was measured.

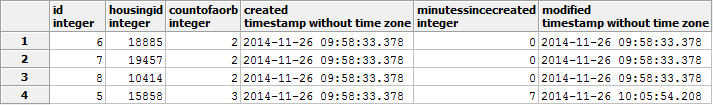
|  |  |
| --- | --- |
| # | Execution Time |
| 1 | 31 |
| 2 | 32 |
| 3 | 31 |
| 4 | 32 |
| 5 | 15 |

What conclusions can be drawn from our execution time results? At best, PostgreSQL query execution benefits slightly with the index on this small amount of data but where indices are really beneficial is on larger data sets. Given the small size of this data, it is easy to think they are not needed, but as more data is loaded, PostgreSQL will begin to rely more and more on indices.

## 2. Advanced Create Table

The following SQL statements were composed to meet the requirements of this question. First the table is created. Next, a trigger function is defined to set the “minutes since created” column based on the created data and ensures the created and modified columns are set appropriately. Thirdly, the trigger function is attached to the honorroll table as a BEFORE INSERT OR UPDATE trigger. Then data is inserted into the table including pre-aggregation of the number of As and Bs a student has (at least 2). An update is made to the table to exercise the trigger’s update functionality. Finally, the data is queried for display.

|  |
| --- |
| CREATE TABLE honorroll  (  id serial PRIMARY KEY,  housingid integer,  countOfAorB integer,  created timestamp,  minutessincecreated integer,  modified timestamp  ); |
| CREATE OR REPLACE FUNCTION f\_honorroll\_calcminsincecreated()  RETURNS TRIGGER  AS $$  BEGIN  IF TG\_OP = 'INSERT' THEN  NEW.created := current\_timestamp;  ELSIF TG\_OP = 'UPDATE' THEN  NEW.created := OLD.created;  END IF;  NEW.minutessincecreated := cast(extract(EPOCH FROM age(current\_timestamp, NEW.created)) / 60 as int);  NEW.modified := current\_timestamp;  RETURN NEW;  END;  $$  LANGUAGE plpgsql; |
| CREATE TRIGGER honorroll\_calcminsincecreated  BEFORE INSERT OR UPDATE ON honorroll  FOR EACH ROW  EXECUTE PROCEDURE f\_honorroll\_calcminsincecreated(); |
| INSERT INTO honorroll (housingid, countofaorb)  SELECT h.id, COUNT(grade) FROM courses c  INNER JOIN housing h ON h.givenname = c.givenname AND h.surname = c.surname  WHERE c.grade IN ('A', 'B')  GROUP BY h.id  HAVING COUNT(grade) > 1; |
| SELECT \* FROM honorroll; |
| UPDATE honorroll SET countofaorb = 3 WHERE id = 5 |



PostgreSQL doesn’t support true calculated columns, rather they are implemented as TRIGGERs on INSERT or UPDATE. As such, the ‘calculation’ is a snapshot based on the last INSERT or UPDATE operation rather than a currently calculated value. Using a VIEW, the calculated column can be recalculated at query time and has the effect of realtime calculation at the expense of the extra CPU cycles needed to recalculate for each query. Additionally, a materialized view could be used to minimize the extra CPU cycles used for recalculation.

## 3. R Performance

A study in performance of summation between the standard for loop, base R’s sum function and a parallelized summation using doParallel and parallel packages was performed. The final paper was uploaded to Rpubs and is available at <http://rpubs.com/dwdii/R-SumInParallel>.

All code associated with this assignment is available in [my DataAcqMgmt GitHub](https://github.com/dwdii/DataAcqMgmt/tree/master/Week13) repository.